ASHESI UNIVERSITY COLLEGE

AUGMENTED REALITY CAPTURE THE FLAG

WUMPINI ALHASSAN HUSSEIN

2014

Applied Project

ASHESI UNIVERSITY COLLEGE

AUGMENTED REALITY CAPTURE THE FLAG

By

WUMPINI ALHASSAN HUSSEIN

Dissertation submitted to the Department of Computer Science,

Ashesi University College

In partial fulfilment of Science degree in Computer Science

MAY 2014

Declaration

I hereby declare that this dissertation is the result of my own original work and that no part of it has been presented for another degree in this university or elsewhere.

Candidate's Signature:……..

Candidate's Name:……

Date:……..

I hereby declare that the preparation and presentation of the dissertation were supervised in accordance with the guidelines on supervision of dissertation laid down by Ashesi University College.

Supervisor's Signature:……..

Supervisor's Name:…….

Date:……..

Acknowledgements

To Dr. Ayorkor Korsah, Mona Lisa, my family and friends: Thank you. For everything.

Abstract

The advent of mobile gaming brought about a new age of how video games are regarded in the context of mobility. Previously, a person would have to own a specialized gaming device that enabled them play on the move. Devices such as the PlayStation Portable and Nintendo's series of GameBoy devices made this a possibility. However, mobile games have always offered a constraint on the type of interactions they offer players. A mobile game is only as enjoyable as the quality of information it displays on the screen and how it displays this information. A new way of viewing the game world would create a new type of interaction for players, and create a new way for them to enjoy mobile games.

This paper reports on the project Augmented Reality: Capture the Flag, which offers a solution to the problem of mundane interactions by using augmented reality as a means of extending the screen of a mobile device. Augmented reality will attempt to reconcile the real world with the game world, creating an illusion of the game world existing in the physical world.

Contents

Table of Contents

Glossary of key terms

Asset	Refers to items that do not come preinstalled with the Unity3D game engine. These items have to be downloaded or purchased from the Unity Asset Store
Augmentation	The superimposition of virtual objects on the real world using augmented reality, giving the illusion that they coexist
Augmented reality	The technology used to achieve augmentation
Bake	Refers to the process of making the game world into a surface that artificial intelligence agents can navigate using different path-finding algorithms
Flag	Refers to the item that will be picked up or captured by the player during gameplay
Marker	Any physical object, or image printed on a flat surface, that serves as a trigger for augmentation
Non-playable character	Refers to the other characters the that exist in the game and perform actions autonomously
Playable character	Refers to the main character of whom the player of the game will assume control
Pose matrix	Refers to how objects are represented in 3D space, including axis orientation

Root motion	Animation driven movement in unity3D. Using root motion, a character will be moved by playing a movement animation and applying a force to the character in the direction of movement
Scene	Refers to all the elements in the game world, including the terrain, characters and lighting
Target	Refers to a marker being used for augmentation. A marker is called a target when the augmented reality application uses it
Terrain	A construct in the Unity3D game engine that represents the landscape or ground where characters will walk
Vector3	A data structure in the Unity3D engine that stores coordinate information in the form of x, y, z.

1. Introduction

This paper is a report on the making of augmented reality capture the flag: An Android game. The game is designed to be a multiplayer game between two people, each person controlling one character and having non-playable characters in support. The point of the game is to collect as many of your opponent's flags as possible before a clock runs out. The game uses augmented reality to extend display from the user's device's screen to a predefined image on a piece of paper. The following sections will give a bit of background to the project and dive deeper into exactly how it will be made possible.

1.1 Background

1.1.1 Mobile Games

Mobile games in general provide very specific types of interactions to players, defined solely by the mechanics of the game. For instance, if someone played a game where they were supposed to put a ball in a hole while avoiding obstacles, the experience would be very similar every time. This experience is limited by the game's mechanics, which is the putting of a ball in a hole while avoiding obstacles. Many other games can be created using the same mechanics but with different sounds and graphics in an attempt to vary the experience. However, the experience very often stays the same. One reason for this phenomenon is the primary output device of a mobile phone: the screen. Using a different output device or extending the screen could create a new type of interaction for the player. Augmented reality provides a way to achieve this.

1.1.2 Augmented Reality

Augmented reality in simple terms "...allows the user to see the real world, with virtual objects superimposed upon or composited with the real world. Therefore, AR supplements reality…" [1]. Essentially this means augmented reality provides a way to superimpose virtual objects such as 3D graphics onto the real world. This creates the illusion of those virtual objects actually existing in our world when in reality they do not. This superimposition can be achieved by projecting the virtual object, or by using a special viewing device that will enable someone to see the object in the real world. These viewing devices could be either Head-Mounted Displays (HMDs) such as Google Glasses or something as simple as a phone's camera [2].

Augmented reality can be split into three main components. These components work in tandem to try to reconcile reality and the virtual objects:

i. **Scene generator:** This is responsible for rendering, which will contain all the virtual objects.
ii. **Tracking system:** The tracking system is responsible for making sure the objects in the real world and the ones in the virtual world are properly aligned to maintain the illusion that they coexist.
iii. **Display:** Displays are predominantly used to view the augmentation. As mentioned earlier, displays could be Head Mounted Displays or a phone [2].

It is appropriate to use augmented reality when the interactions needed cannot necessarily be duplicated using a mouse or keyboard or monitor, or any combination of those three. When creating augmented reality applications, it is important to make the merger of real and virtual objects as seamless as possible. This can be done by choosing the right form of Augmentation for the right situations. There are three main ways to augmentation [3]:

i. **Augment the user:** Here, the user wears or carries a device to obtain information about a physical object. The method poses a registering problem: the ability to precisely match real world objects to their corresponding electronic information [3].
ii. **Augment the physical object:** The physical object is changed by embedding input, output, or computational devices within it or on it [3].
iii. **Augment the environment surrounding the user and the object:** Here, neither the user nor the environment is affected directly. Different devices provide and collect information from the surrounding environment, displaying information about the user's interactions with them [3].

In most Android games, characters appear on the screen of the playing devices and the user interacts with them by touching her screen or using a peripheral input device. Using augmented reality, it is possible to extend the display of characters onto a physical surface, making the device a medium for which the player could view and interact with characters as if they exist in the real world. An example would be the ball in the game

mentioned earlier, instead of being on the player's screen, appearing on the physical floor.

1.2 Objectives

The objective of this report is to examine the processes and outcomes of the making of the project: Augmented Reality Capture the Flag. To recap, it is an Android game that uses augmented reality in an attempt to create a new type of interaction for players. The objective of the game is for a player to outwit his or her opponent by seizing a flag at their home base. While doing this, the player should use all means necessary to stop his/her opponent from reaching and capturing his/her flag. The person with the highest number of captured flags at the end of a timed round wins the game. The entire game world will be rendered on an augmented reality marker (a simple image) which in simple terms tells the game what to render. A high level analogy of how a marker works is in a conditional statement in programming. If the player's device recognizes that particular image, it should render the game world onto it.

The prototype game documented in this report will serve as a fully playable template from which a bigger game using augmented reality will be made. It also serves as a source of entertainment for people, exposing them to augmented reality, the type of interaction it provides and its possible applications. Also, I have gained considerable skills in using augmented reality and in creating peer-to-peer applications using the AllJoyn software development kit.

Creating this game required among other things javascript and C#, having no knowledge of C# at the beginning of the project. Also, this project

helped me gain experience with the Unity3D game engine. The Unity3D game engine is vast therefore the skills gained concern these parts of the entire engine:

- **The physics engine:** This is the part of the Unity3D game engine responsible for all physics simulations and everything to make game objects look and act like they would in the real world with properties such as gravity, mass and drag.
- **The animation system (Mecanim):** This part of the engine deals with all animations that can be applied either to playable characters or non-playable characters.
- **The particle system:** This part is responsible for rendering things such as smoke, fire and dust.
- **The Navigation system:** This system makes it possible for non-playable characters to find their way around the game world.
- **GUI system:** This is the graphical user interface system responsible for rendering all user interface elements such as menus.
- **Input system:** Responsible for handling player input, mainly through scripts written either in C# or JavaScript.
- **Lighting system:** This system is responsible for illuminating game objects in a game scene.
- **Sound system:** This system is responsible for creating sound based on events in the game and playing background music.

Finally, in creating non-playable characters that will serve as extra opponents, I had to create Artificial Intelligence agents that roamed the game autonomously and reacted to the player's character in various ways.

This part of the project required a fusing of the knowledge of the systems stated above and many smaller systems required to create AI agents in the Unity3D engine.

1.3 Outline of Report

This report will proceed to examine the design of the game, after which the implementation, including techniques and technologies used will also be discussed. After, some tests and results will be analyzed leading to a conclusion and some recommendations.

2. Design

The game is designed to be a multiplayer game which can be played by joining one of two teams. Through years of observation of people who play games and after examining a cross-section of individuals in my vicinity, I realized there were two main types of players: those who planned their moves and every action, and those who didn't. This advised my decision in picking the names of the two teams. A team comprises of a playable character and some artificial intelligence agents. The name of the first team is Team Intoms. "Intoms" is a word used in Twi and more in Pidgin English to represent chance or haphazardness. The second team is called Team Stra. "Stra" is short for strategy and is used by most Ghanaians who speak Pidgin English to mean an elaborate and systematic plan. The purpose of using these two words (Intoms and Stra), is to help define a target audience for the game and help people identify with at least one of the teams.

2.1 Audience

This game is targeted towards mobile gamers looking for a different type of interaction. It is suitable for people of all ages who have a general sense of how to use a smartphone, although it will be slightly more enjoyable to people who speak or understand Pidgin English and can relate to a team. Steps were taken to ensure that the game contained no violence or strong language or replicable actions that may bring harm to individuals.

2.2 Requirements

The design of the game were bound by certain standards and constraints. These partition the project into smaller parts and guided the entire process. These standards or requirements are listed below under functional and non-functional requirements.

2.2.1 Functional requirements

The following requirements describe what the game should do:

2.2.1.1 Interface requirements

i. The game should be able to successfully identify and render all 3d elements onto the special marker the player will use as a target.

ii. After successfully rendering 3d elements, other elements essential to gameplay such as input buttons and the timer should be visible to the player at the start of the game.

iii. The game should display the number of flags the user has captured so far and the number of points they are worth, in a non-obtrusive way.

iv. The input buttons available to the player should be an analog stick, a run button and an attack button. These buttons should also be non-obtrusive. The player's screen will be seamlessly split in two halves, the left being for the analog stick and the right being for other buttons. The analog stick should remain hidden until the player touches the left part of the screen. When the player lifts their finger from the screen,

it should fade out. Also, it should appear on any part of the left partition the player touches. This is to ensure that the player's vision is not obscured unnecessarily. The buttons on the right side should always be visible.

v. The player should be able to see the amount of health left of their character.

Before enumerating the game's core functional requirements, here is a recap of how the game works:

2.2.1.2 Game rules

The game is timed for one minute and the timer starts when a player hits play. At the start of the game, both players have one objective: To capture all the flags of the opponent. This will be done while avoiding the opponent's Artificial Intelligence agents who have the objective of depleting the health of their opponents, hence stopping them from capturing all the flags.

Each flag will have a specific point value. Points will be calculated by summing the point values of each flag collected. The winner when the time runs out will be the player with the maximum number of points or the player who is first to capture all flags. The total number of flags for each team is five. A winning state could be realized in the event that an opponent's health reaches zero. In this instance, the opposing player, irrespective of the number of points accrued, will be the winner.

2.2.1.3 Core functional requirements:

i. Each player should have some amount of health that depletes based on the number times their opponent's Artificial Intelligence interacts with them.

ii. The game should start with the player's character at a point in the map far away from the objective points. This is to enforce some level of difficulty. Because the game is timed, the player will need to be wise about which flag to pick up first, since different flags have different point values.

iii. The actions corresponding to the input buttons should be applied to rendered 3d objects on the marker, in a similar way it would have been applied on a normal screen.

iv. The timer should start counting down from the time the game starts and should stop the game when the time expires.

v. Once the player's character walks over a flag, it should disappear and its point values computed and added to the player's score.

vi. Some flags will be guarded by obstacles. The player will be armed with an object that can be used to inflict damage on the obstacles, eventually destroying them and gaining access to the flag.

vii. The game should have Artificial Intelligence agents who will be responsible for distracting the player. These agents will shoot projectiles at the player that have a potential of depleting the player's health. The player's character is

disabled when health reaches zero and the game ends. At this point, the player will have the option of closing the game or restarting it.

2.3 Non-Functional requirements

The following requirements will specify how the game should behave. They represent the quality attributes of the game.

i. Capacity: Considering data charges and the availability of internet, the game should be as small as possible, without compromising its quality. To achieve this, the game should not occupy more than 60mb of space on the user's device.

ii. Availability: When development of the game is complete, it will be available for download from the Play Store on Android devices.

iii. Reliability: The game should start every time it is launched by the user. In the unlikely event the game does not start, it should not be a factor of the game's poor design. For example it should be a reason like low battery power on the user's device and not that the game's design does not permit it to run on battery power less than 15%.

iv. Security: The game should not act in a way that will compromise the security of the user's device. It only needs access to the device's camera during gameplay, and forgoes access afterwards.

v. Usability: The first time a user opens the game, it should be obvious to them how to play it and navigate all the screens. They should be able to do that at their own pace. When the user leaves the game for a long period and launches it again, it should be easy for them to remember how to maneuver the screens and play the game. If a user makes an error, the game should make it immediately recoverable. For example, mistakenly pressing restart during a game is an error which the game makes recoverable by asking for confirmation. Finally, the game should be pleasant to play.

2.4 Use case and Scenario

When a user plays the game, they interact with many systems directly and indirectly. Direct interaction is between the user and their playing device and also between the user and AR marker. These systems in turn interact with the AR system itself and all its components.

In terms of use cases of the game, there is only one possible use case which is actually playing the game. There are very few possible scenarios the player can and will encounter, the first being the winning scenario. Winning consists of being able to successfully collect all flags. The second possible scenario will be losing the game. This consists of either not being able capture all flags within the given timeframe, or being deactivated by one of the AI agents. After either losing or winning, the player has the chance to either restart or end the entire game.

3 Implementation

The game was made with the unity3D game engine using JavaScript and C# for programming the gameplay, with the aid of plugins (known in Unity lingo as *assets*) to create the Graphical User Interface for controlling the player's movement. The game uses augmented reality as a means of providing a new type of interaction for the players. The augmented reality development kit used is known as Vuforia, and is developed by Qualcomm. Finally, the Software Development kit that was used to implement multiplayer functionality is called AllJoyn, also developed by Qualcomm.

The general approach taken to complete this project was to first of all identify all the tools and assets needed. The choice of tools for the project was motivated by several reasons; the first factor being how feasible it would be to combine these tools into one product. The second factor was the pricing of these tools, since not all Software Development Kits come for free. The third factor was the fact that these tools represent emerging technologies, and learning about them would be a wonderful addition to my skillset.

After all these considerations, unit tests of each piece of technology were conducted to help me assess exactly how well they worked individually, they're capabilities, deficiencies and possible areas of concern. Integration tests were then conducted with the software development kits and the game engine to ensure they could work well together.

3.1 Technologies

This will provide an in-depth view of the technologies used in this project

i. Vuforia AR SDK:

The Vuforia SDK is arranged in components. These components include:

a. The camera, which is responsible for ensuring that the captured images of the target are efficiently tracked and parsed to the image tracker component.
b. The image converter, which is responsible for ensuring that the images being tracked are converted to the right format before being parsed to the tracker.
c. The Tracker, which contains a computer vision algorithm that is responsible for finding and keeping track of real-world objects. The algorithms used for detection change based on the type of target being tracked that is either normal image targets, 3d targets or virtual buttons. The results of tracking are stored in an object which is accessible programmatically. The tracker is capable of tracking multiple targets at the same time.
d. The video background renderer, which is responsible for rendering the image stored from the image tracker.
e. The Application code, which is all the code the application developer will write to perform three key tasks. These tasks include polling the tracked object to see if it has changed or

there is a new target, using input data to update the application and finally to render the augmented graphics.

f. The device databases, which contain data about the image targets. A device database can be created from an online target manager. The image that will be used as an image target is uploaded and converted in a database. It is then downloaded and included in the application as a local database, referred to every time an instance of the application runs.

g. The cloud databases which act as an alternative to a local database, a cloud database stores information on the cloud and has to be queried every time the application runs.

h. The user defined targets, which allows for targets to be created at run time using the camera's current image. This way, instead of the developer packaging the target with the application, the user can create his or her own target.

i. The word targets, which enable the SDK to track either whole words or just characters. This can be used when the application being built has to recognize specific words in order perform actions. A word is recognized only if it belongs to a word list, which is the equivalent of the device database for image targets. A word list can be extended to contain words that the developer chooses. When recognizing characters, any character including numbers can be recognized.

Below is an image detailing the work flow of the Vuforia SDK

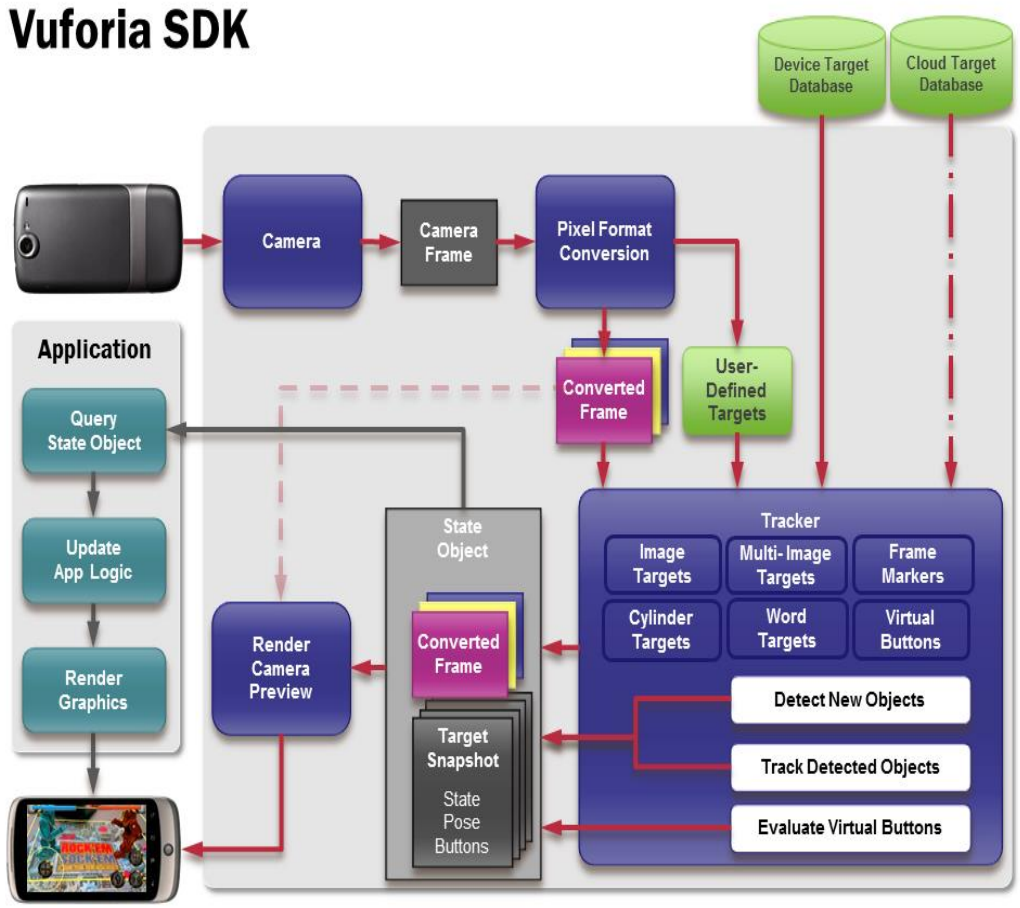

Fig. 3.1: Vuforia Software Development Kit workflow [4]

Below is an image of the application development process using the Vuforia SDK

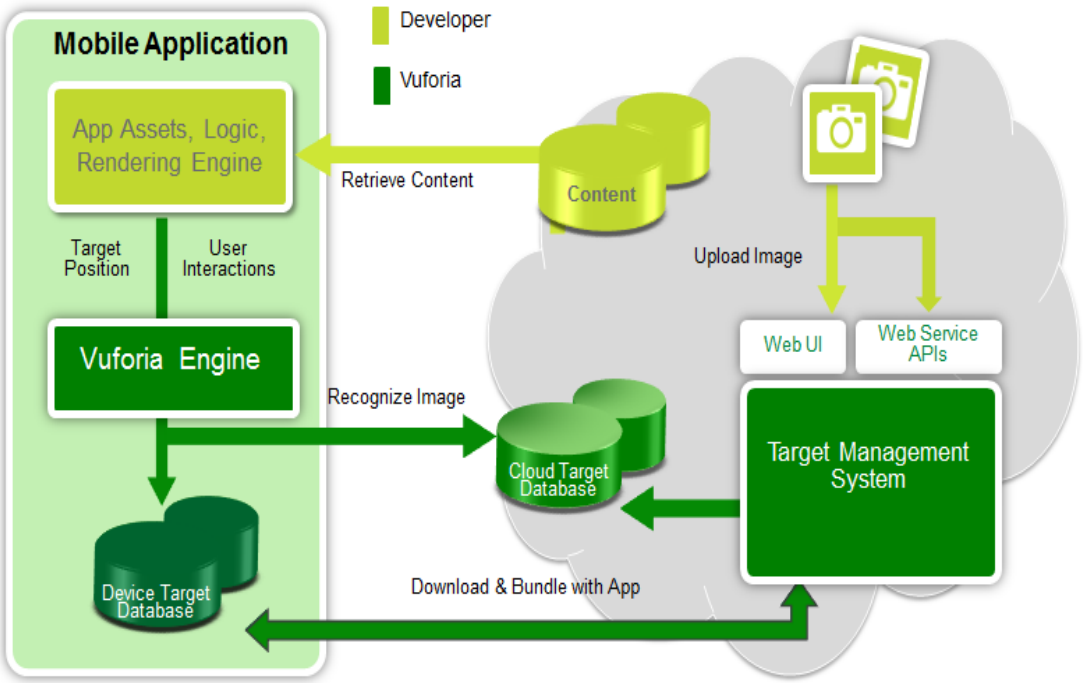

Fig 3.2: Application development process using Vuforia Software Development Kit [5]

ii. AllJoyn SDK

This SDK is responsible for enabling the multiplayer part of the game. AllJoyn is also a product of Qualcomm that enables the creation of peer-to-peer experiences over devices of different types and different operating systems. This means that it can enable a Microsoft Windows computer talk to an Android phone or an iPhone. This works when these devices are in proximity to each other, meaning they are connected via the same wireless networks or one of them serves as the source of the wireless

network. This also works over Bluetooth networks even though these have a relatively shorter range that Wi-Fi.

Before two applications can talk to each other via the AllJoyn SDK, one of them must advertise and the other must discover the advertisement. This is achieved by using an AllJoyn BusAttachment. This object enables applications to connect to the AllJoyn SDK and to be able to advertise, discover and communicate. The creation of the BusAttachment has to be done programmatically, after which an auto-generated unique name is created and assigned to that BusAttachment. This unique name is referred to as a *well-known name* in AllJoyn lingo and is a unique identifier of that instance of AllJoyn that other instances can discover and connect to. These names are alpha-numeric and can contain special symbols. If two applications have the same well-known name and a third wants connect to one of them, it will not know which one to connect to. This is illustrated by the image below:

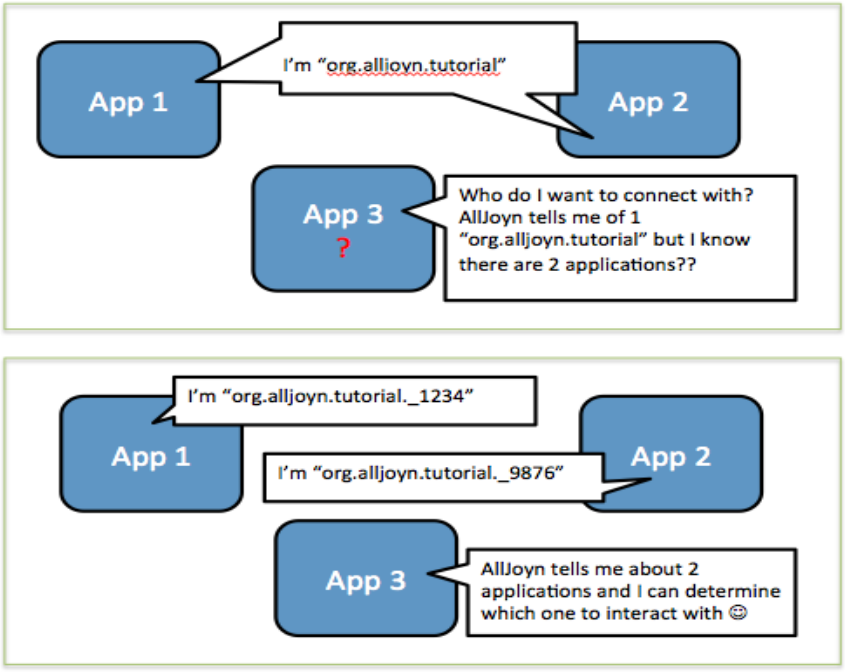

Fig. 3.3: Discovery process of an AllJoyn enabled application [6]

As stated earlier, the first step for an AllJoyn application is to connect to AllJoyn via the BusAttachment. After this, a medium of communication is determined and a unique identifier is chosen, or the unique name the developer provides that instance of the application is used. After this, the application takes one of two routes; the first being that it acts as an advertiser or in more common terms a host. In this mode, a group can be created from which other applications can join via the well-known name. After the group is created, it is then advertised with its well-known name so that other applications searching for an instance running with that name will find it. If a connection is established, the two applications will be able to exchange information.

The second route makes the application act like a client in the sense that, it looks for a host to connect to. It does this by searching nearby applications to see if they have a particular well-known name. If any of

them do, that application is connected to and information can be exchanged from then. Actually, applications built with AllJoyn can advertise and search for a well-known name at the same time. This enables the peer-to-peer communication and gets rid of the client server paradigm.

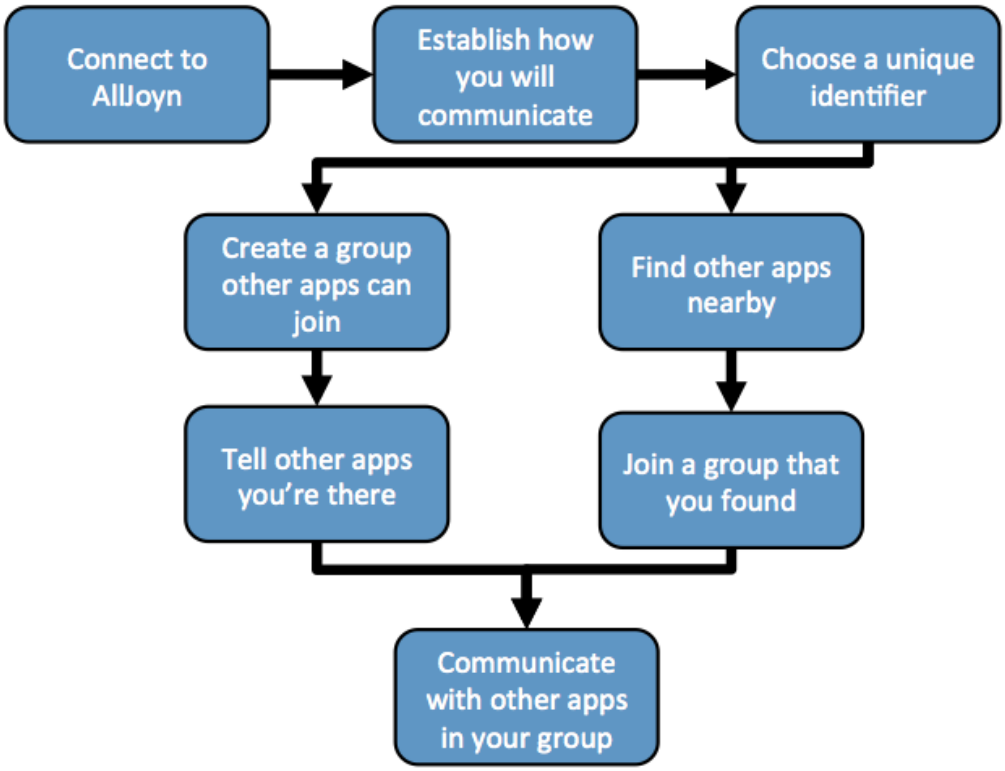

Fig. 3.4: The activity flow of an AllJoyn enabled application [7]

3.2 Approach

In developing the game, the first things done as stated earlier were unit tests. These tests included testing different parts of the various development kits that were going to be used. The purpose of this was to familiarize myself with them and to ensure that they actually did work. After doing these tests, I proceeded to doing some integration tests

between development kits and the game engine. This was to ensure that all tools worked well together and also to help me familiarize myself with the workflow of developing such a complex system with multiple components, which is not very typical of the mobile game development process using Unity3D.

I started out by designing the image targets for both teams of the game. In the spirit of being a source of entertainment, the targets were made as comical as possible just so they could appeal visually to the player.

Team stra:

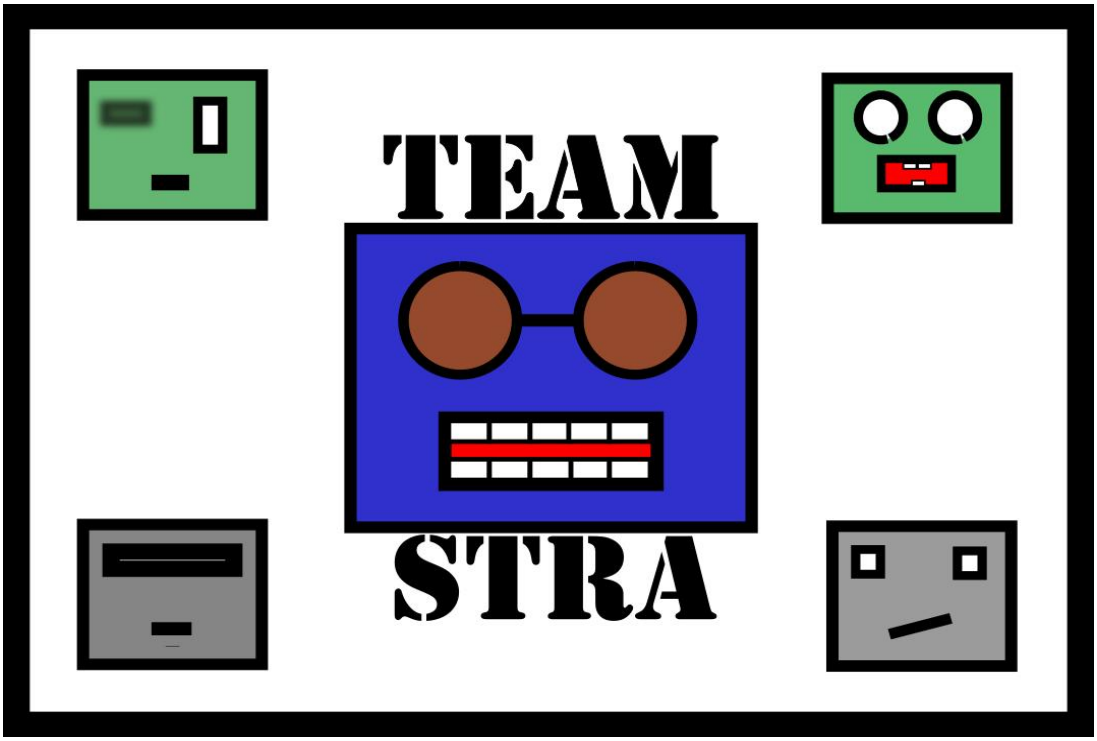

Fig. 3.5: The marker for team stra

Team intoms:

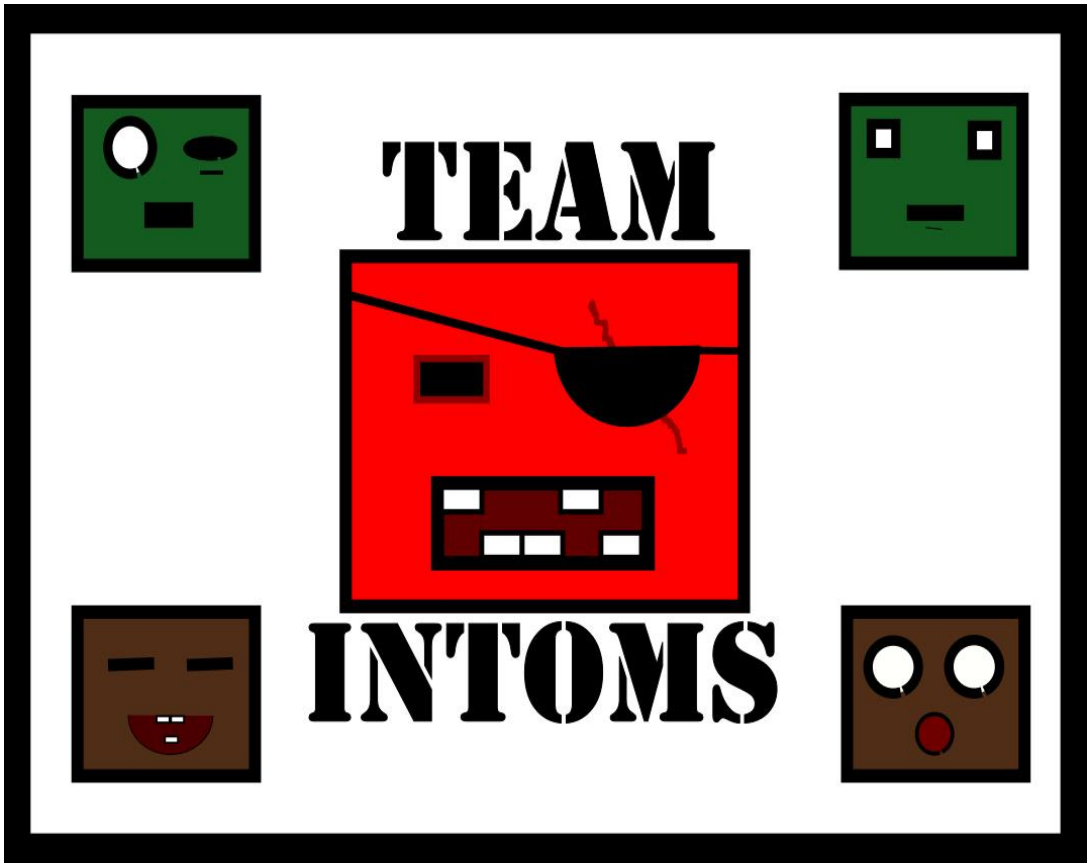

Fig. 3.6: The marker for team Intoms

Following the Vuforia workflow, I uploaded the image targets and downloaded them as databases, which I then added to the game engine through the Vuforia SDK plugin for Unity3D. This made the database accessible to the game.

The next step was the creation of terrains that would be rendered on the image targets and the placement of playable and non-playable characters. Once this was done, the next step was to actually script the characters and gameplay. This part was split mainly in three, one part for the main

playable character's movement and actions, the other part for the AI agents and the third part for the game's state management.

The main character's movement and actions were a particular difficulty given I had no prior experience with game development. Since the character is a humanoid, I started out by trying to map player input with application of forces to particular parts of the character's body in order to move it. This took many weeks and did not work due to my general lack of understanding and experience. Later, I swapped the humanoid character with a sphere, which was now my main character. With the sphere, I was able to move it around by applying a constant force in the direction of the player's input. When the player pressed another direction, I applied drag to the sphere before applying force in that direction. This gave the sensation of the ball having weight.

Not long after using the ball, I discovered *root motion*, a construct of the Unity3D game engine that allowed me to move a humanoid character by playing animations (all through scripting). This meant that if the player pressed move in a particular direction, I would simply have to turn the character in that direction and play the walk animation until the player released the button or changed the direction, at which point I would repeat the steps again. After this great discovery, I once again switched the ball for the humanoid character, which could now move about in the game world.

Moving about in the game world also presented a unique challenge: Both Unity and the Vuforia software development kit have their own pose matrices. To make these systems work together for this project required

that both systems used the same pose matrix, in order for screen input to be translated into real world movement. Screen input here represented information on the Unity pose matrix, which had to be manipulated in order to move a character by its corresponding value in the Vuforia pose matrix.

In the Vuforia pose matrix, the x-axis is to the right, the y-axis going up along the marker and the z-axis going into and out of the marker. On the other hand, the Unity pose matrix has the z-axis up and along the marker and the y-axis moving in and out of it, x-axis stays the same. The problem this created was that, if the player moved the playable character forward on the screen via the joystick, instead of moving in the z-axis, it moved in the y-axis causing the playable character to walk upwards towards the screen. Also, the Vuforia pose matrix offers coordinates relative to the playing device's camera. This adds another level of complexity because now the character will always move relative to the orientation of the device.

The way I went about solving this was to find a way to transform the pose matrix of Vuforia to match the pose matrix of Unity. My first attempt involved explicitly swapping the y and z axes of either matrix. This involved me creating a Vector3 variable with the coordinates from user input, and swapping the positions of the y and z axes. This method did not work well as it did not account for the fact that movement was relative to the camera. After exploring many more ways of solving this problem including saving the coordinates as matrices and transforming them, I found a solution that worked.

By changing the world center mode in the settings of the camera accessed through the Unity3D engine, I was able to correct this problem. I changed it from being set to CAMERA, to SPECIFIC_TARGET. By making this change, I was telling Vuforia that it should make all movement relative to a specific target (which I set as my game world's floor).

The next step after this was to script the AI agents. This was a brute-force way of creating AI which again given my lack of experience, was the only feasible way. The AI agents' capabilities were limited and lacked complex behavior. The agents had a field of view and followed the main character if he moved within this field of vision. That particular agent in pursuit of the main character would then try to catch up with him and attack him, unless the main character was able outrun it and make the distance between them greater than a particular threshold I set. The creation of the AI agent was particularly interesting as it also required me to *bake* the game scene. This process of baking basically ensured that the character would be able to interact with objects in the scene and be able to navigate the scene using algorithms such as the A* algorithm.

The third part of the scripting was the game state manager, which also included the input manager and timer for the game. The state manager is responsible for starting the screen and presenting the player with a home screen before the game. It controls user input on the menu level and manages which scene of the game is shown depending on what the user choses on the home screen. Once the user choses to start the main game, the timer and input manager start.

The timer counts down from when the game starts and stops the game when the time expires. It then presents the user with the option of quitting the game or restarting it. The input manager listens for player input and performs actions as directed. On a deeper level, it is also responsible for rendering the buttons onto the screen of the phone and hiding buttons when they are not needed. This is to ensure that the user always has a clear screen from which to view the terrain.

Below is a list of screenshots of the game scene, buttons and user interface. These were taken from the test device used during development, which is a Samsung Galaxy Note II.

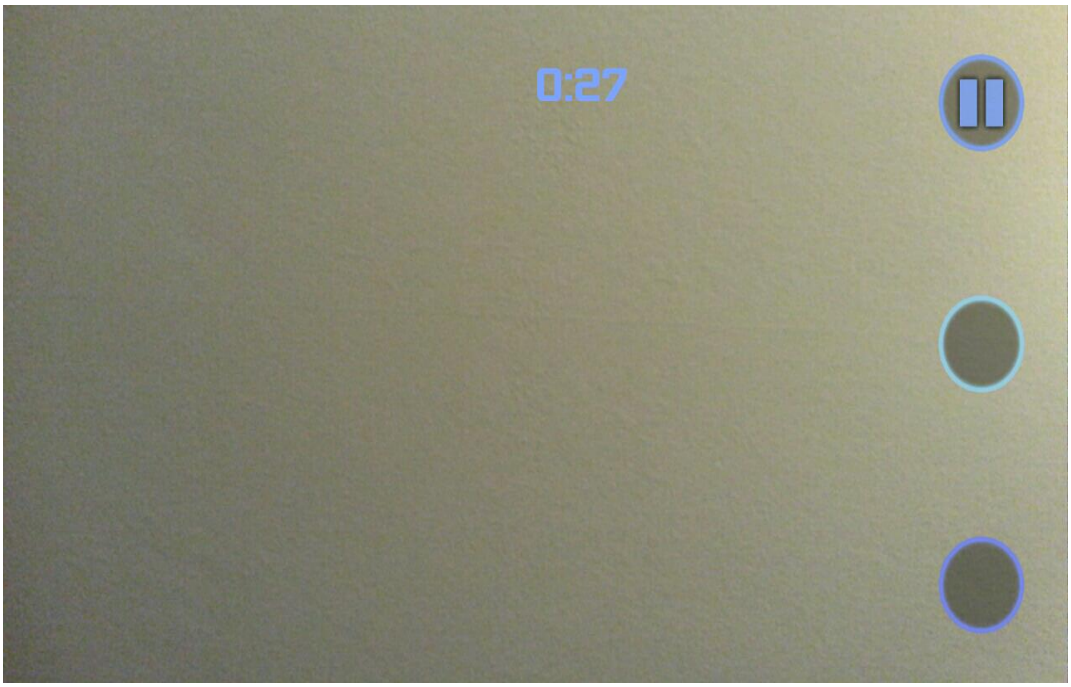

Fig. 3.7: This is the default view of the game when nothing has started and the target is not being tracked.

To the extreme right of the phone's screen are three buttons. One of them is very obviously a pause button and the other two are attack and run

buttons respectively. The attack button has a light blue outline and the run button has a dark blue outline. The positioning of the buttons was very deliberate because I did not want it to be in a place where it would occupy the screen or cause the user to have to move fingers across the screen. Also, all buttons are transparent save their outlines. This is also to ensure that while not pressing the buttons, they stay as unobtrusive as possible.

Fig. 3.8: This is when the target has been found, is being tracked, and the augmentation starts.

In this view, the game scene has been rendered onto the marker. Also being rendered is the main character.

Fig. 3.9: A closer look at how detailed the rendering is.

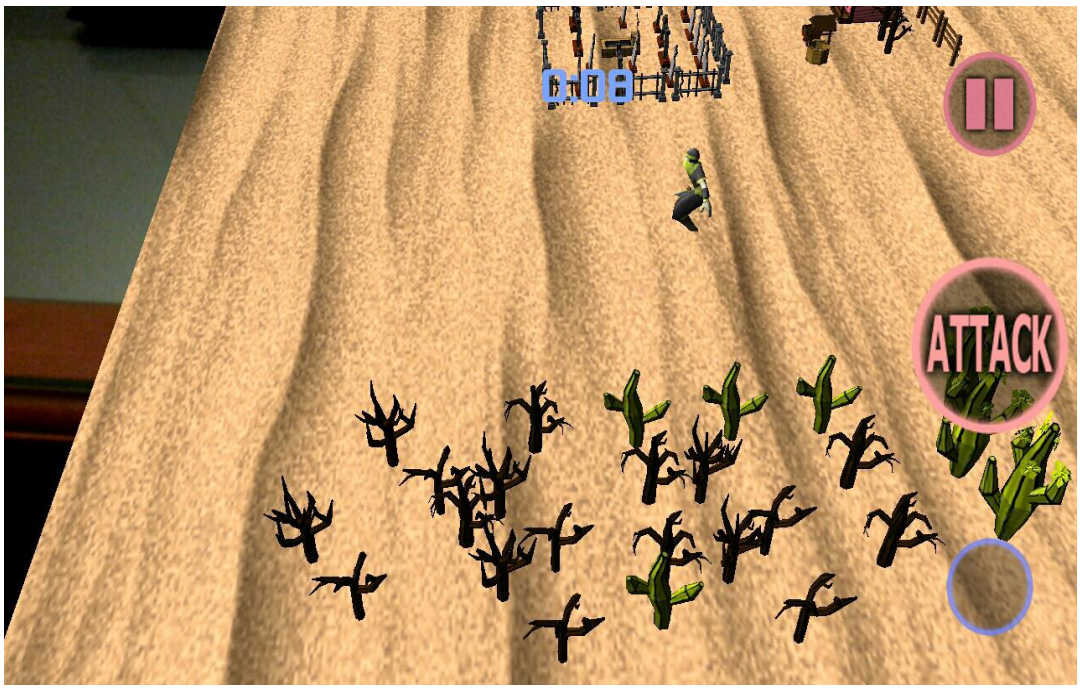

Fig. 3.10: When the attack button is pressed.

When the attack button is pressed, it increases in size and stays like that as long as the player's finger is on the screen. If the player's finger is

lifted, it will revert to its original state. This also applies to the run button below.

Fig. 3.11: When run button is pressed.

The buttons demonstrated above are the action buttons. For movement, the character can be controlled via a joystick that pops up at any part of the left screen the player touches. It then disappears when the player lifts their finger. This way, like the other buttons, it is as unobtrusive to the player as much as possible.

Fig. 3.12: When the left part of the screen is pressed, a joystick appears.

The level of implementation of this game does not include multiplayer functionality. This functionality will be introduced in future implementations. Currently, the player is able to play using the same game rules, only with artificial intelligence agents as opponents.

4 Test and Result

Throughout the development process, tests were carried out on different components as they were developed. These tests ranged from simple tests to usability tests. As the development process went on, these tests revealed problems I did not foresee at the beginning of the project.

In terms of usability tests, I conducted five tests with five different subjects. These subjects comprised of three boys and two girls. One of the boys was from Ireland, the other from Lithuania and the last from England. They ranged between the ages of 20 to 23 and had experience with mobile games, none with augmented reality. The two girls were from Sweden and England and were 19 and 21 respectively. They also had experience with mobile games and none with augmented reality.

While conducting these tests, a number of issues surfaced:

i. The movement joystick's visibility: The fact that the movement joystick remained hidden until the left part of the screen was touched meant that the users did not know how to move the character. All users took a while before touching the screen, not because they knew what they were doing but because they were just trying something. After the joystick appeared, it became apparent how to use it to move the character. General comments from the users about this issue pointed towards the fact that it would be very helpful to have an instruction screen just before the game starts.

i. The fact that the rendered graphics are not interactive: During the tests, a few users stretched out their hands in an attempt to

touch the main character or an object in the game world, and were often disappointed when nothing happened. Comments from users about this suggested that it would be a good idea to have some form of physical interaction with the game instead solely relying on the buttons on the screen for input.

Other problems faced during testing included performance issues while using Unity's built in terrains. With these terrains, it would be possible for me to create beautiful landscapes with features such as forests, mountains, waterfalls and swamps. Unfortunately, this weighed heavily on the performance on the game, delaying the game's starting time by a very noticeable number of seconds. I found out later after doing a lot of research there were ways of optimizing terrains to improve performance. At this point however, I had already swapped out the terrain for a simple cube which I flattened and widened. This meant that my terrain was now very flat and uneven with no features. While this is not necessarily a bad thing, having a more realistic terrain would improve the look and feel of the game.

5 Conclusions and Future implementations

This report has examined the background and motivation of the project Augmented Reality Capture the Flag. It went further to discuss the tools used giving reasons for the choices of tools. It also discussed the approach for implementing the project, detailing problems faced and how these problems were solved. Finally, it presented tests conducted and the problems with usability and performance they exposed, and how they would be addressed.

5.1 Future implementations

Features will be added to the current version of this project. Also, it will be extended to accommodate more teams. The notable features that will be added will include the following:

i. Multiplayer functionality: This will allow two players play the game where they will try to outwit each other. This functionality can be achieved using the AllJoyn software development kit described in the Implementation section above. Alternatively, the Unity3D game engine offers Master Server, which can be used to implement multiplayer functionality. Further reading on this can be done at [8].

ii. Left handed gameplay: This will enable left handed players customize the game's controls to have the joystick on the right hand side of the screen, and the action buttons on the left.

iii. Better artificial intelligence: This will increase the abilities of the artificial intelligence agents to enable them perform

special actions such as cover finding when being attacked by other agents or players. Also, the advanced artificial intelligence agents will have temperaments that reflect the team they belong.

iv. Tutorial screen: This screen will briefly appear just before the game starts to display instructions on moving the playable character and moving performing actions.

v. Physical interaction with game world and players: This will enable the game world to react to physical touches from a player. For instance if the player touches the playable character, the character might run away from the player's finger. This can be achieved using the virtual buttons functionality of the Vuforia software development kit.

References

[1] R. T. Azuma, “A Survey of Augmented Reality,” in *Teleoperators and Virtual Environments 6*, Malibu, 1997.

[2] R. Silva, J. C. Oliveira and G. A. Giraldi, “Introduction to Augmented Reality,” National Laboratory for Scientific Computation, Petropolis, 2003.

[3] W. E. Mackay, “Augmented Reality: Linking real and virtual worlds,” Department of Computer Science: Université de Paris-Sud, Orsay-Cedex, 1998.

[4] Qualcomm Connected Experience, “Vuforia Developer Resources,” Qualcomm, 2014. [Online]. Available: https://developer.vuforia.com/resources/dev-guide/vuforia-ar-architecture. [Accessed 29 March 2014].

[5] Qualcomm Connected Experience, “Developing with Vuforia,” Qualcomm, 2014. [Online]. Available: https://developer.vuforia.com/resources/dev-guide/getting-started. [Accessed 29 March 2014].

[6] AllSeen Alliance, “Well-Known Name (Uniquename Alias),” AllSeen Alliance, 2014. [Online]. Available: https://allseenalliance.org/well-known-name-uniquename-alias. [Accessed 29 March 2014].

[7] AllSeen Alliance, “AllJoyn Application Process,” AllSeen Alliance, 2014. [Online]. Available: https://allseenalliance.org/alljoyn-application-process. [Accessed 29 March 2014].

[8] Unity Technologies, “Master Server,” Unity3D, 10 October 2013. [Online]. Available: http://docs.unity3d.com/Documentation/Components/net-MasterServer.html. [Accessed 15 April 2014].